Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА
Институт Информационных Технологий
Кафедра Промышленной Информатики

ПРОЦЕДУРНОЕ ПРОГРАММИРОВАНИЕ

Тема лекции «РАЗЛИЧИЯ СИНТАКСИСА С И С++. ТЕРНАРНЫЕ ОПЕРАТОРЫ. ССЫЛКИ. ФАЙЛЫ»

Лектор Каширская Елизавета Натановна (к.т.н., доцент, ФГБОУ ВО "МИРЭА - Российский технологический университет") e-mail: liza.kashirskaya@gmail.com

Лекция № 8

НЕКОТОРЫЕ РАЗЛИЧИЯ СИНТАКСИСА С И С++

С++ — язык общего назначения и задуман для того, чтобы настоящие программисты получили удовольствие от самого процесса программирования.

Язык программирования С++ задумывался как язык, который будет:

- лучше и современней языка С;
- поддерживать абстракцию данных;
- поддерживать объектно-ориентированное программирование.
- содержать большую и расширяемую стандартную библиотеку.

НЕКОТОРЫЕ РАЗЛИЧИЯ СИНТАКСИСА С И С++

За исключением второстепенных деталей, он практически содержит язык С как подмножество. Язык С++, в свою очередь, является расширением языка С, его надмножеством. Программист может структурировать свою задачу, определив новые типы, которые точно соответствуют понятиям предметной области задачи. Такой метод построения программы обычно называют абстракцией данных. Информация о типах содержится в некоторых объектах типов, определенных пользователем. С такими объектами можно работать надежно и просто даже в тех случаях, когда их тип нельзя установить на стадии трансляции. Программирование с использованием таких объектов обычно называют объектно-ориентированным. Если этот метод применяется правильно, то программы становятся короче и понятнее, а сопровождение их упрощается.

Можно сказать, что Си и C++ сосуществуют между собой. Когда в 2011 году вышел новый стандарт языка C++ — C++11, вместе с ним вышел и стандарт языка Си — C11.

C++ - это тот же Си, но с некоторыми удобными упрощениями. Часто можно слышать споры на тему: писать на Си или на C++? При этом существует расхожее мнение о том, что есть два стиля написания программ: стиль C и стиль C++. Они противопоставляются друг другу. C++ ассоциируется с ООП (объектно-ориентированным программированием), а чистый Си - с ПОП (процедурно-ориентированным программированием). ООП и ПОП также противопоставляются.

На самом деле, все, что есть нового в C++, уже было в Си. Только в C++ это записывается чуть по-другому. Так как мы с вами начали изучение процедурно-ориентированного программирования с модификации языка C++, а не Си, необходимо сформулировать главные различия этих языков, а вернее, модификаций одного и того же языка. Мы не будем рассматривать элементы объектно-ориентированного программирования, а покажем лишь некоторые различия в операторной части.

Все заголовочные файлы стандартной библиотеки языка C++ не содержат расширения .h.

Например:

```
#include<iostream>
#include<vector>
#include<algorithm>
```

Заголовочные файлы из стандартной библиотеки языка Си можно использовать в языке C++, но их имена изменились - в начало имени файла добавилась буква "c", а расширение ".h" исчезло. То есть при желании использовать функции, которые в языке Си определены в заголовочных файлах stdio.h или math.h, их требуется подключать следующим образом:

```
#include<cstdio>
#include<cmath>
```

Язык C++ содержит обширную стандартную библиотеку.

Главные составляющие операторной части библиотеки следующие:

1) ввод-вывод, он описан в заголовочных файлах iostream, fstream и др.;

2) работа со строками, она описана в файле string.

Имена (функций, переменных) в языке C++ можно разделять на «пространства имен» для удобства - чтобы могли существовать функции и переменные с одинаковыми именами в разных «пространствах имен».

Пространство имен объявляется так:

```
namespace my_namespace
{
   // Описание функций, переменных, классов
   int var;
};
```

Для доступа к переменной var из пространства имен **my_namespace**

нужно писать
my_namespace::var

Можно также использовать инструкцию:
using namespace my_namespace;

Тогда все имена из пространства имен **my_namespace** можно использовать без указания имени пространства имен (просто писать **var**).

Вся стандартная библиотека находится в пространстве имен **std**, поэтому нужно либо писать

std::cout << a << std::endl;

для вывода переменной *a*.

Или написать в начале программы

using namespace std;

и тогда можно будет просто писать

cout << a << endl;

Для стандартного ввода-вывода в языке C++ используется заголовочный файл **iostream**

В нем объявлены объекты **cin** для ввода с клавиатуры и
cout для вывода на экран.

Чтобы считать со стандартного ввода значения переменных a, b, c, нужно написать:
std::cin >> a >> b >> c;

Для вывода на экран этих переменных нужно написать:
std::cout << a << b << c;

Для разделения значения переменных пробелами нужно выводить строку из одного пробела между ними:

std::cout << a << " " << b << " " << c;

Чтобы вывести конец строки, нужно вывести стандартный объект **endl**:

std::cout << std::endl;

Можно не писать **std::**, как вы знаете, если дать инструкцию

using namespace std

в начале программы.

Комментарии в тексте программы

В языке Си допускались только многострочные комментарии. Начало комментария обозначалось символами /*, конец - символами */.

Пример:

/* Это комментарий.
он может занимать несколько строк */

В языке C++ появились однострочные комментарии - они отмечаются символами // и продолжаются до конца строки:

int n; // Размер считываемого массива

В языке Си для объявления констант используются директивы препроцессора **#define.**

Например:
***#define N* 100**

Это низкоуровневый и опасный механизм. Например, объявленную таким образом константу на самом деле можно переопределить:
#define *N* 1000

В языке C++ появились константные выражения, которые нужно использовать вместо **#define**:
const int *N* = 100;

Одним из наиболее важных различий между Си и C++ является тот факт, что в Си функция, объявленная следующим образом:

```
int f();
```

ничего не говорит о своих параметрах. Это означает, что функция может иметь параметры или не иметь их вовсе. В отличие от этого, в C++ объявленная таким образом функция не имеет параметров. Иными словами, в C++ следующие два объявления эквивалентны:

```
int f();
int f(void);
```

В C++ ключевое слово **void** является факультативным. Многие программисты на C++ включают **void** в качестве средства, улучшающего читаемость программы и указывающего, что у функции нет параметров.

Функция main()

Первым исполненным оператором становится первый оператор функции main(). При запуске программы управление всегда передается функции main(). При попытке запустить программу без данной функции компилятор выдаст сообщение об ошибке.

```
#include < stdio.h >
int main ( )
{
   printf ( "Let's start smth else\n" );
   return 0;
}
```

Директивы

Первая строка, с которой начинается верхний код, является директивой. Эта строка – директива препроцессора компилятору.

#include<stdio.h> является директивой препроцессора, и, в отличие от оператора, который дает компьютеру указание что-либо сделать, директива указывает компилятору.
Файл, включаемый с помощью директивы **#include**, – заголовочный файл.

Функции ввода и вывода

В языке Си используются функции **printf()** и **scanf()** для вывода и ввода соответственно. Для того, чтобы эти функции работали, надо подключить библиотеку **<stdio.h>**.

Для работы с динамической памятью вместо функций **malloc** и **free** языка Си в языке C++ введены операторы **new** и **delete**. Использование функций языка Си для работы с динамической памятью не рекомендуется в языке C++.

По большей части C++ представляет собой надстройку над стандартным ANSI Си, и фактически все программы на Си являются также программами и на C++. Тем не менее, между этими языками имеется несколько различий, и наиболее важные из них обсуждаются здесь.

Управляющие последовательности языка Си

Управляющая последовательность означает, что символ \ «управляет» интерпретацией следующих за ним символов последовательности.

Управляющая последовательность	Значение
\t	Горизонтальная табуляция
\n	Переход на новую строку
\v	Вертикальная табуляция
\r	Возврат каретки
\”	Кавычки
\’	Апостроф
\\	Обратная дробная черта
\ddd	Восьмеричный код символа
\xddd	Шестнадцатеричный код символа

Пример.

```
#include <stdio.h>

int main ( ) {
    int i;
    for ( i = 1; i <= 10; i ++ ) {
        printf( "%d", i );
    }
    return 0;
}
```

В кавычках указывается **формат вывода**: %(буква) обозначает тип формата замещающего текста.
%s задает строку,
%d-целое число,
а %c-символ.

Пример.

```c
#include <stdio.h>

int main ( ) {
    int n = 1;
    int m = 0;
    printf ( "Please enter a nubmer of integers. In the end, enter 0: " );
    while ( n != 0 ) {
        scanf ( "%d", &n );
        m += n;
    }
    printf ( "The sum equals: &d", m );
    return 0;
}
```

Пример.

```c
#include <stdio.h>

int main ( ) {
    int n = 1;
    int m = 0;
    printf ( "Enter a number of integers. In the end, enter 0: " );
    do {
      scanf ( "%d", &n );
      m += n;
    } while ( n != 0 );
    printf ( "The sum equals: %d", m );
    return 0;
}
```

Пример.

```c
#include <stdio.h>

int main ( ) {
    int n;
    printf ( "Input a number: " );
    scanf ( "%d", &n );
    if ( n > 5 ) {
      printf ( "Your number is more than 5" );
    }
    return 0;
}
```

Пример.

```c
#include <stdio.h>

int main ( ) {
    int n;
    printf ( "Enter a number: " );
    scanf ( "%d", &n );
    if ( n > 5 ) {
        printf ( "Your number is more than 5" );
    } else {
        printf ( "Your number is less than 5" );
    }
    return 0;
}
```

При запуске программы из командной строки ей можно передавать дополнительные параметры в текстовом виде. Например, следующая команда

ping -t 5 google.com

будет отправлять пакеты на адрес google.com с интервалом в 5 секунд. Здесь мы передали программе ping три параметра: «-t», «5» и «google.com», которые программа интерпретирует как задержку между запросами и адрес хоста для обмена пакетами.

Параметры командной строки в C++

В программе эти параметры из командной строки можно получить через аргументы функции main при использовании функции main в следующей форме:

int main(int argc, char* argv[]) { /* ... */ }

Первый аргумент содержит количество параметров командной строки. Второй аргумент — это массив строк, содержащий параметры командной строки. Т.е. первый аргумент указывает количество элементов массива во втором аргументе.

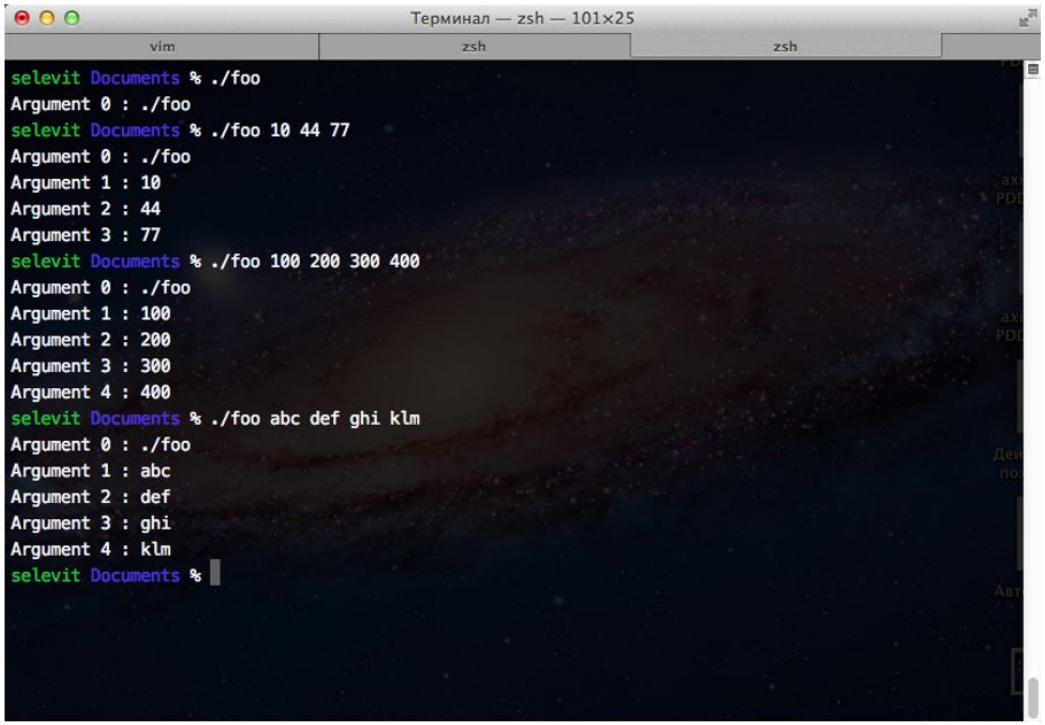
Первый элемент массива строк (`argv[0]`) всегда содержит строку, использованную для запуска программы (либо пустую строку). Следующие элементы (от 1 до `argc - 1`) содержат параметры командной строки, если они есть. Элемент массива строк `argv[argc]` всегда должен содержать 0.

Пример

```
#include <iostream>
using namespace std;

int main(int argc, char *argv[])
{
   for (int i = 0; i < argc; i++) {
      // Выводим список аргументов в цикле
      cout << "Argument " << i << " : " << argv[i] << endl;
   }
   return 0;
}
```

Откройте командную строку и запустите оттуда скомпилированную программу.



Для получения числовых данных из входных параметров, можно использовать функции atoi и atof.

УСЛОВНЫЙ ТЕРНАРНЫЙ ОПЕРАТОР

Условный (тернарный) оператор (обозначается как `?:`) является единственным тернарным оператором в языке C++, который работает с 3-мя операндами. Из-за этого его часто называют просто ***«тернарный оператор»***.

Оператор	Символ	Пример	Операция
Условный	?:	c ? x : y	Если c — ненулевое значение (true), то вычисляется x, в противном случае — y

Оператор `?:` предоставляет сокращенный способ (альтернативу) ветвления if/else.

Стейтменты if/else:

```
if (условие)
    выражение;
else
    другое_выражение;
```

Можно записать как:

```
(условие) ? выражение : другое_выражение;
```

Обратите внимание, операнды условного оператора должны быть выражениями (а не стейтментами).

Например, ветвление if/else, которое выглядит следующим образом:

```
if (условие)
    x = значение1;
else
    x = значение2;
```

Можно записать как:

```
x = (условие) ? значение1 : значение2;
```

Большинство программистов предпочитают последний вариант, так как он читабельнее.

УСЛОВНЫЙ ТЕРНАРНЫЙ ОПЕРАТОР

Давайте рассмотрим еще один пример. Чтобы определить, какое значение поместить в переменную `larger`, мы можем сделать так:

```
if (x > y)
    larger = x;
else
    larger = y;
```

Или вот так:

```
larger = (x > y) ? x : y;
```

Обычно, часть с условием помещают внутри скобок, чтобы убедиться, что приоритет операций корректно сохранен и так удобнее читать.

Помните, что оператор `?:` имеет очень низкий приоритет, из-за этого его следует записывать в круглых скобках.

Например, для вывода `x` или `y`, мы можем сделать следующее:

```
if (x > y)
    std::cout << x;
else
    std::cout << y;
```

Или с помощью тернарного оператора:

```
std::cout << ((x > y) ? x : y);
```

УСЛОВНЫЙ ТЕРНАРНЫЙ ОПЕРАТОР

Давайте рассмотрим, что произойдет, если мы не заключим в скобки весь условный оператор в вышеприведенном случае. Поскольку оператор `<<` имеет более высокий приоритет, чем оператор `?:`, то следующий стейтмент (где мы не заключили весь тернарный оператор в круглые скобки, а только лишь условие):

```
1 std::cout << (x > y) ? x : y;
```

Будет обрабатываться как:

```
1 (std::cout << (x > y)) ? x : y;
```

Таким образом, в консольном окне мы увидим `1` (true), если `x > y`, в противном случае — выведется `0` (false).

Совет: Всегда заключайте в скобки условную часть тернарного оператора, а лучше весь тернарный оператор.

Условный тернарный оператор — это удобное упрощение ветвления if/else, особенно при присваивании результата переменной или возврате определенного значения. Но его не следует использовать вместо сложных ветвлений if/else, так как в таких случаях читабельность кода резко ухудшается и вероятность возникновения ошибок только растет.

Правило: Используйте условный тернарный оператор только в тривиальных случаях.

Условный тернарный оператор вычисляется как выражение

Стоит отметить, что условный оператор вычисляется как выражение, в то время как ветвление if/else обрабатывается как набор стейтментов. Это означает, что тернарный оператор **?:** может быть использован там, где **if/else** применить невозможно, например, при инициализации константы:

```
1 bool inBigClassroom = false;
2 const int classSize = inBigClassroom ? 30 : 20;
```

Здесь нельзя использовать if/else, так как константы должны быть инициализированы при объявлении, а стейтмент не может быть значением для инициализации.

ССЫЛКИ В C++. ОТЛИЧИЕ ОТ УКАЗАТЕЛЕЙ

Давайте вспомним, что мы знаем про указатели, так как эти два элемента языка – ссылки и указатели – буквально, что называется, ходят рядом.

Каждая переменная или объект хранит данные в определенной ячейке в памяти компьютера. Каждый раз, создавая новую переменную, мы создаем новую ячейку в памяти, с новым значением для неё. Чем больше ячеек, тем больше компьютерной памяти будет занято.

Адрес в памяти компьютера - это число, к которому мы можем получить доступ.

Указатель - это тот же адрес в памяти, по которому мы получаем переменную и её значение.

Чтобы работать с указателями, необходимо воспользоваться двумя специальными символами: **&** и ***.

Пример использования указателя:

int t = 237; // Простая переменная

int *p; // Создание указателя, который принимает лишь //адрес другой переменной

p = &t; // Устанавливаем адрес нашей первой переменной

Переменные **t** и ***p** будут равны числу **237**, так как оба ссылаются на одну ячейку. Сам же компьютер на вычислении обеих переменных потратит меньше усилий, ведь обе переменные ссылаются на одно и то же.

Пример использования указателя:

```
int x; i
int *y = &x; // От любой переменной можно взять
//адрес при помощи операции взятия адреса "&".
//Эта операция возвращает указатель
int z = *y; // Указатель можно разыменовать при
помощи операции
// разыменовывания "*". Эта операция
возвращает тот объект, на который указывает
указатель
```

Теперь по поводу ссылок. **Ссылки** — это то же самое, что и указатели, но с другим синтаксисом и некоторыми другими важными отличиями, о которых речь пойдёт дальше. Следующий код ничем не отличается от предыдущего, за исключением того, что в нём фигурируют ссылки вместо указателей:

```
int x;
int &y = x;
int z = y;
```

Если слева от знака присваивания стоит ссылка, то нет никакого способа понять, хотим мы присвоить правую часть самой ссылке или объекту, на который она ссылается. Поэтому такое присваивание всегда присваивает правую часть объекту, а не ссылке. Но это не относится к инициализации ссылки: инициализируется, разумеется, сама ссылка. Поэтому после инициализации ссылки нет никакого способа изменить её саму, т.е. ссылка всегда постоянна (но не её объект).

Ссылка в C++ - это альтернативное имя объекта. Ссылку можно понимать как безопасный вариант указателя. При этом ссылки имеют особенности, отличающие их от указателей.

1. При объявлении ссылка создается обязательно на уже существующий объект данного типа. Ссылка не может ссылаться "ни на что".
2. Ссылка от её объявления до её исчезновения указывает на один и тот же адрес.
3. При обращении к ссылке разыменование происходит автоматически.
4. Адрес ссылки - это адрес исходного объекта, на который она указывает.

Объявление ссылок очень похоже на объявление указателей, только вместо звёздочки * пишется амперсанд &.

При объявлении ссылка обязана быть инициализирована.

```
int &x; // недопустимо!
int &x = veryLongVariableName; // допустимо.
```

Теперь x - это альтернативное имя переменной veryLongVariableName

```
int A[10];
int &x = A[5]; // Ссылка может указывать на элемент массива
x++;          // то же, что A[5]++;
x = 1;       // то же, что A[5] = 1;
```

Ссылки против указателей

Ссылки часто путаются с указателями, но вот три основных различия между ссылками и указателями.

- У вас не может быть ссылок NULL. Вы всегда должны иметь возможность предположить, что ссылка связана с законной частью хранилища.
- Когда ссылка инициализируется объектом, ее нельзя изменить, чтобы ссылаться на другой объект. Указатели могут указывать на другой объект в любое время.
- Ссылка должна быть инициализирована, когда она создана. Указатели могут быть инициализированы в любое время.

Ссылки и указатели схожи между собой, так как оба в качестве значения имеют лишь адрес некого объекта.

Указатель хранит адрес ячейки, и если мы захотим изменить значение этой ячейки, то нам придется выполнить операцию «разыменования»:

```
float some = 391; // Простая переменная
float *u = &some; // Указатель на переменную
*u = 98; // Изменение значения переменной
```

Ссылки против указателей

В ссылках такого понятия нет, так как, меняя ссылку, вы автоматически меняете и переменную. Ссылки напрямую ссылаются к переменной, поэтому их синтаксис проще:

```
char symbol = 'A'; // Простая переменная
char &ref = symbol; // Создание ссылки на переменную
// Поскольку мы ссылаемся на переменную, то можем её использовать
// как отдельно взятую переменную
cout << ref << endl; // Вывод символа "A"
ref = 'C'; // Изменение на символ "C"
```

Использование ссылок и указателей оправдано в случае передачи данных в функции или же в объекты. Также данные технологии отлично подойдут для передачи большого объема данных в ходе выполнения программы.

Создание ссылок на C++

Представьте себе, что имя переменной – это метка, прикрепленная к местоположению переменной в памяти. Тогда вы можете считать ссылку вторым ярлыком, прикрепленным к этой ячейке памяти. Таким образом, вы можете получить доступ к содержимому переменной через имя исходной переменной или через ссылку. Например, предположим, что мы имеем следующий пример:

```
int i = 17;
```

Мы можем объявить ссылочные переменные для **i** следующим образом:

```
Int &r = i;
```

Создание ссылок на C++. Пример

```
#include <iostream>
using namespace std;

int main ()
{
// объявляем простые переменные
int i;
double d;
// объявляем ссылочные переменные
int &r = i;
double &s = d;

i = 5;
cout << "Value of i : " << i << endl;
cout << "Value of i reference : " << r << endl;

d = 11.7;
cout << "Value of d : " << d << endl;
cout << "Value of d reference : " << s << endl;
return 0;
}
```

Считайте & в этих объявлениях **ссылкой** . Так им образом, прочитайте первое объявление как «r - целочисленная ссылка, инициализированная i», и прочитайте второе объявление как «s – ссылка двойной точности, инициализированная d». То есть здесь используются ссылки на int и double.

Результат выполнения программы:

```
Value of i : 5
Value of i reference : 5
Value of d : 11.7
Value of d reference : 11.7
```

Создание ссылок на C++

Ссылки обычно используются для списков аргументов функций и возвращаемых значений функции.

Следующие два важных вопроса, связанные с ссылками на C ++, которые должны быть понятны программисту - это использование ссылок в качестве параметров и в качестве возвращаемых значений.

Ссылки как параметры

С ++ поддерживает передачу ссылок в качестве функционального параметра.

Передача аргументов по ссылке

Когда мы изучали функции, то рассмотрели общие моменты создания функций. Разумеется, существует некоторое число нюансов, которые будут описаны далее. Первым является **передача аргументов по ссылке**.

Ранее в примере аргументы передавались в функцию по значению. Это значило, что в функции мы работали с копиями этих объектов, а если размер объектов слишком большой или важна память, очевидно, что лишние расходы ее на копии ни к чему. Другим аспектом, заслуживающим внимания, является то, что иногда от функции требуется вернуть больше одного значения, но синтаксис Си не позволяет этого сделать. И тут на помощь приходят ссылки.

Ссылка представляет собой механизм, который не создает копии, а позволяет работать с самой переменной, указанной в качестве аргумента.

Пример программы, возводящей число в степень:

```
#include <stdio.h>

void power ( int *, int );

int main ( )
{
    int n = 2;
    int s = 7;
    power ( &n, s );
    printf ( "The result is: %d", n );
    return 0;
}

void power ( int *number, int stepen )
{
    int k = *number;
    for ( ; stepen > 1; stepen -- ) {
        *number *= k;
    }
}
```

В листинге данного кода функция power( ) принимает адрес, который присваивается локальной переменной-указателю. В теле функции происходит изменение значения по адресу, содержащемуся в указателе. Однако по факту это адрес переменной **n** из функции main( ), следовательно, меняется и значение **n**.

ВАЖНО. Передавать по адресу можно лишь переменные, если вы вместо переменной передадите какое-то значение, к примеру, в нашем случае вместо &n передать n, то вызов этой функции приведет к ошибке.

Ссылка как возвращаемое значение

Вы можете вернуть ссылку из функции C ++, как и любой другой тип данных.

Чего нельзя делать со ссылкой

Возврат из функции ссылки на автоматически созданный объект (локальную переменную) приводит к появлению "битых ссылок", значение которых непредсказуемо.

Также синтаксис C++ не позволяет создавать указатели на ссылки и массивы ссылок.

РАБОТА С ФАЙЛАМИ

Большинство компьютерных программ работают с файлами, и поэтому возникает необходимость создавать, удалять, записывать, читать, открывать файлы. Что же такое файл? **Файл – именованный набор байтов, который может быть сохранен на некотором накопителе.** Ясно, что под файлом понимается некоторая последовательность байтов, которая имеет своё, уникальное имя, например файл.txt. В одной директории не могут находиться файлы с одинаковыми именами. Под именем файла понимается не только его название, но и расширение, например: file.txt и file.dat — разные файлы, хоть и имеют одинаковые названия. Существует такое понятие, как полное имя файлов – это полный адрес к директории файла с указанием имени файла, например: D:\docs\file.txt.

РАБОТА с ФАЙЛАМИ

Итак, файлом является способ хранения информации на физическом устройстве. Файл — это понятие, которое применимо ко всему — от файла на диске до терминала.

В C++ отсутствуют операторы для работы с файлами. Все необходимые действия выполняются с помощью функций, включенных в стандартную библиотеку. Они позволяют работать с различными устройствами, такими, как диски, принтер, коммуникационные каналы и т. д. Эти устройства сильно отличаются друг от друга. Однако файловая система преобразует их в единое абстрактное логическое устройство, называемое потоком.

Текстовый поток — это последовательность символов. При передаче символов из потока на экран часть из них не выводится (например, символ возврата каретки, перевода строки).

Двоичный поток — это последовательность байтов, которые однозначно соответствуют тому, что находится на внешнем устройстве.

ОРГАНИЗАЦИЯ РАБОТЫ С ФАЙЛАМИ СРЕДСТВАМИ Си

Объявление файла:

FILE *идентификатор;

Пример. FILE *f;

Открытие файла:

fopen(имя физического файла, режим доступа);

Режим доступа — строка, указывающая режим открытия файла и тип файла.

Типы файла: бинарный (b); текстовый (t).

Режимы доступа к файлам:

- `"r"` — открыть файл для чтения (файл должен существовать);
- `"w"` — открыть пустой файл для записи; если файл существует, то его содержимое теряется;
- `"a"` — открыть файл для записи в конец (для добавления); файл создается, если он не существует;
- `"r+"` — открыть файл для чтения и записи (файл должен существовать);
- `"w+"` — открыть пустой файл для чтения и записи; если файл существует, то его содержимое теряется;
- `"a+"` — открыть файл для чтения и дополнения, если файл не существует, то он создаётся.

Например,

```
f = fopen(s, "wb");
k = fopen("h:\ex.dat", "rb");
```

Запись в файл:

fwrite(адрес записываемой величины, размер одного экземпляра, количество записываемых величин, имя логического файла);

Например,

fwrite(&dat, sizeof(int), 1, f);

Чтение из файла:

fread(адрес величины, размер одного экземпляра, количество считываемых величин, имя логического файла);

Например,

fread(&dat, sizeof(int), 1, f);

Закрытие файла:

fclose(имя логического файла);

Пример 1. Заполнить файл некоторым количеством целых случайных чисел.

/* Заполнить файл некоторым количеством целых случайных чисел. */

```
/* Dev-C++ */
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
 FILE *f; int dat;
 srand(time(0));
 int n=rand()%30 + 1;
 cout << "File name? ";
 char s[20];
 cin.getline(s, 20);
 f=fopen(s, "wb");
 for (int i=1; i<=n; i++)
 { dat = rand()%101 - 50;
   cout << dat << " ";
   fwrite(&dat, sizeof(int), 1, f);
 }
 cout << endl;
 fclose(f);
 return EXIT_SUCCESS;
}
```

Пример 2. Найти сумму и количество целых чисел, записанных в бинарный файл.

```
/* Найти сумму и количество целых чисел, записанных в бинарный файл. */
/* Dev-C++ */
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
 FILE *f;
 int dat, n=0, sum=0;

 cout << "File name? ";
 char s[20];
 cin.getline(s, 20);
 f=fopen(s, "rb");
 while (fread(&dat, sizeof(int), 1, f))
      {n++;
       cout << dat << " ";
       sum+=dat;
      }
 cout << endl;
 cout << "sum: " << sum << "; number: " << n << endl;
 fclose(f);
 system("PAUSE");
 return EXIT_SUCCESS;
}
```

Пример 3. Поместить в файл **n** записей, содержащих сведения о кроликах, содержащихся в хозяйстве: пол (m/f), возраст (в месяцах), масса.

```
/* Поместить в файл n записей, содержащих сведения о кроликах, содержащихся в хозяйстве:
   пол (m/f), возраст (в мес.), масса. */

/* Dev-C++ */
#include <cstdlib>
#include <iostream>

using namespace std;

struct krolik {char pol; int vozrast; double massa;};

int main()
{
 FILE *f; krolik dat; int n;
 cout << "File name? ";
 char s[20];
 cin.getline(s, 20);
 f=fopen(s, "wb");
 cout << "How many rabbits? "; cin >> n;
 for (int i=1; i<=n; i++)
 { cout << "What sex " << i << "th rabbit? "; cin >> dat.pol;
   cout << "How old " << i << "th rabbit? "; cin >> dat.vozrast;
   cout << "What is the mass of the " << i << "th rabbit? "; cin >> dat.massa;
   fwrite(&dat, sizeof(krolik), 1, f);
 }
 fclose(f);
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Пример 3 (продолжение). В бинарном файле хранятся сведения о кроликах, содержащихся в хозяйстве: пол (m/f), возраст (в мес.), масса. Найти наиболее старого кролика. Если таких несколько, то вывести информацию о том из них, масса которого больше.

/* В бинарном файле хранятся сведения о кроликах, содержащихся в хозяйстве: пол (m/f), возраст (в мес.), масса. Найти наиболее старого кролика. Если таких несколько, то вывести информацию о том из них, масса которого больше.*/

```
/* Dev-C++ */
#include <cstdlib>
#include <iostream>

using namespace std;

struct krolik {char pol; int vozrast; double massa;};

int main()
{
 FILE *f; krolik dat, max; int n;
 cout << "File name? ";
 char s[20];
 cin.getline(s, 20);
 f=fopen(s, "rb");
 fread(&dat, sizeof(krolik), 1, f);
 max=dat;
 while (fread(&dat, sizeof(krolik), 1, f))
 {if (dat.vozrast>max.vozrast) max=dat;
  else if (dat.vozrast==max.vozrast&&dat.massa>max.massa) max=dat;}
 cout << "The oldest rabbit has a sex " << max.pol << ", age " << max.vozrast << " and mass " << max.massa << endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Форматированный файловый ввод-вывод

1) Функции fgetc() и fputc() позволяют соответственно осуществить ввод-вывод символа.
2) Функции fgets() и fputs() позволяют соответственно осуществить ввод-вывод строки.
3) Функции fscanf() и fprintf() позволяют соответственно осуществить форматированный ввод-вывод и аналогичный соответствующим функциям форматированного ввода-вывода, только делают это применительно к файлу.

Файловый ввод-вывод с использованием потоков

Для работы с файлами необходимо подключить заголовочный файл <fstream>. В <fstream> определены несколько классов и подключены заголовочные файлы <ifstream> — файловый ввод и <ofstream> — файловый вывод.

Библиотека потокового ввода-вывода:

fstream

Связь файла с потоком вывода:
ofstream имя логического файла;

Связь файла с потоком ввода:
ifstream имя логического файла;

Открытие файла:
имя логического файла.open(имя физического файла);

Закрытие файла:
имя логического файла.close();

Простой файловый ввод-вывод

Предположим, программа должна выполнять запись в файл. Понадобится предпринять следующие действия.

1. Создать объект **ofstream** для управления выходным потоком.

2. Ассоциировать этот объект с конкретным файлом.

3. Использовать объект так же, как нужно было бы использовать **cout**.

Единственным отличием будет то, что **вывод направляется в файл вместо экрана**.

Чтобы достичь этого, нужно начать с подключения заголовочного файла **fstream**. Его подключение в большинстве, хотя и не во всех реализациях, автоматически подключает файл **iostream**, поэтому явное подключение **iostream** не обязательно. Затем нужно объявить объект типа **ofstream**:

```
ofstream fout; // создание объекта fout типа ofstream
```

Именем объекта может быть любое допустимое в C++ имя, такое как fout, outFile, cgate или didi.

Затем этот объект нужно ассоциировать с конкретным файлом. Это можно сделать с помощью метода open(). Предположим, например, что требуется открыть файл jar.txt для вывода. Это можно было бы сделать следующим образом:

```
fout.open("jar.txt"); // связывание fout с файлом jar.txt
```

Эти два шага (создание объекта и ассоциация файла с ним) можно совместить в одном операторе, используя другую форму:

```
ofstream fout("jar.txt"); // создание объекта fout и его ассоциирование
//с файлом jar.txt
```

После того, как все это сделано, fout (или любое другое выбранное вами имя) можно будет использовать таким же образом, как и cout. Например, если требуется поместить слова Dull Data в этот файл, это можно сделать следующим образом:

```
fout << "Dull Data";
```

Действительно, поскольку в ostream — содержится ofstream, можно применять все методы ostream, включая разнообразные операции вставки, а также методы форматирования и манипуляторы. Класс ofstream использует буферизованный вывод, поэтому при создании объекта типа ofstream, такого как fout, программа выделяет пространство для выходного буфера. Если вы создадите два объекта ofstream, программа создаст два буфера — по одному для каждого объекта. Объект ofstream, подобный fout, накапливает выходные данные программы байт за байтом, а затем, когда буфер заполняется, передает его содержимое в файл назначения. Поскольку система спроектирована для передачи данных более крупными порциями, а не побайтно, буферизованный подход значительно увеличивает скорость передачи данных из программы в файл.

Такое открытие файла для вывода создает новый файл, если файла с указанным именем не существует. Если же файл с этим именем существовал ранее, то действие по его открытию усекает его так, чтобы вывод начинался в пустой файл.

Действия для выполнения чтения из файла во многом подобны тем, которые необходимы для выполнения записи в файл.

1. Создать объект ifstream для управления входным потоком.
2. Ассоциировать этот объект с конкретным файлом.
3. Использовать объект так же, как нужно было бы использовать cin.

Простой файловый ввод-вывод

Шаги для чтения из файла похожи на те, которые нужно выполнить для записи в файл. Для начала, конечно, нужно подключить заголовочный файл fstream. Затем необходимо объявить объект ifstream и ассоциировать его с именем файла. Для этого можно использовать два оператора или же один:

```
ifstream fin; // создать объекта fin типа ifstream
fin.open("jellyjar.txt"); // открытие файла jellyjar.txt для чтения
// Один оператор
ifstream fis("jamjar.dat") ; // создание объекта fis и его ассоциирование
   // с файлом jamjar.txt
```

Затем объект fin или fis можно использовать почти так же, как cin.

Например, можно использовать следующий код:

```
char ch;
fin >> ch; // считывание символа из файла jellyjar.txt
char buf[80];
fin>> buf; // считывание слова из файла
fin.getline(buf, 80); // считывание строки из файла
string line;
getline(fin, line); // считывание из файла в строковый объект
```

Ввод, как и вывод, также буферизуется, поэтому создание объекта ofstream, такого как fin, создает входной буфер, которым управляет объект fin. Как и в случае вывода, буферизация обеспечивает гораздо более быстрое перемещение данных, чем при побайтной передаче.

Соединение с файлом закрывается автоматически, когда объекты ввода и вывода уничтожаются, например, по завершении программы. Кроме того, соединение с файлом можно закрыть явно, используя для этого метод close():

```
fout.close (); // закрытие соединения вывода с файлом
fin.close (); // закрытие соединения ввода с файлом
```

Файловый ввод-вывод с использованием потоков

Файловый ввод/вывод аналогичен стандартному вводу/выводу, единственное отличие – это то, что ввод/вывод выполнятся не на экран, а в файл. Если ввод/вывод на стандартные устройства выполняется с помощью объектов cin и cout, то для организации файлового ввода/вывода достаточно создать собственные объекты, которые можно использовать аналогично операторам cin и cout.

Например, необходимо создать текстовый файл и записать в него строку «Работа с файлами в C++». Для этого необходимо проделать следующие шаги:

создать объект класса ofstream;

связать объект класса с файлом, в который будет производиться запись;

записать строку в файл;

закрыть файл.

Почему необходимо создавать объект класса ofstream, а не класса ifstream? Потому, что нужно сделать запись в файл, а если бы нужно было

Шаг 1. Создаём объект для записи в файл:

ofstream /*имя объекта*/; // объект класса ofstream

Назовём объект – fout.

Вот что получится:

```
ofstream fout;
```

Для чего нам объект? Объект необходим, чтобы можно было выполнять запись в файл. Уже объект создан, но не связан с файлом, в который нужно записать строку.

Шаг 2. Связываем объект с файлом.

Через операцию «точка» получаем доступ к методу класса open(), в круглых скобочках которого указываем имя файла. Указанный файл будет создан в текущей директории с программой. Если файл с таким именем существует, то существующий файл будет заменен новым. Итак, файл открыт, осталось записать в него нужную строку. Делается это так:

```
fout << "Работа с файлами в C++"; // запись строки в файл
```

Используя операцию передачи в поток совместно с объектом fout, строка «Работа с файлами в C++» записывается в файл. Так как больше нет необходимости изменять содержимое файла, его нужно закрыть, то есть отделить объект от файла.

```
fout.close(); // закрываем файл
```

Итог – создан файл со строкой «Работа с файлами в C++».

Шаги 1 и 2 можно объединить, то есть в одной строке создать объект и связать его с файлом. Делается это так:

```
ofstream fout("cppstudio.txt"); // создаём объект класса ofstream и
//связываем его с файлом cppstudio.txt
```

Объединим весь код и получим следующую программу:
// file.cpp: определяет точку входа для консольного приложения.

```
#include "stdafx.h"
#include <fstream>
using namespace std;

int main(int argc, char* argv[])
{
   ofstream fout("cppstudio.txt"); // создаём объект класса ofstream для записи и связываем его с файлом cppstudio.txt
   fout << "Работа с файлами в C++"; // запись строки в файл
   fout.close(); // закрываем файл
   return 0;
}
```

Осталось проверить правильность работы программы, а для этого открываем файл cppstudio.txt и смотрим его содержимое, должно быть — «Работа с файлами в C++».

Файловый ввод-вывод с использованием потоков

Для того чтобы прочитать файл, понадобится выполнить те же шаги, что и при записи в файл с небольшими изменениями:
создать объект класса ifstream и связать его с файлом, из которого будет производиться считывание;
прочитать файл;
закрыть файл.
// file_read.cpp: определяет точку входа для консольного приложения.

```
#include "stdafx.h"
#include <fstream>
#include <iostream>
using namespace std;

int main(int argc, char* argv[])
{
   setlocale(LC_ALL, "rus"); // корректное отображение Кириллицы
   char buff[50]; // буфер промежуточного хранения считываемого из файла текста
   ifstream fin("cppstudio.txt"); // открыли файл для чтения

   fin >> buff; // считали первое слово из файла
   cout << buff << endl; // напечатали это слово

   fin.getline(buff, 50); // считали строку из файла
   fin.close(); // закрываем файл
   cout << buff << endl; // напечатали эту строку
   return 0;
}
```

В программе показаны два способа чтения из файла: первый – с использованием операции передачи в поток, второй – с использованием функции getline(). В первом случае считывается только первое слово, а во втором случае считывается строка длинной 50 символов. Но так как в файле осталось меньше 50 символов, то считываются символы до последнего включительно. Обратите внимание на то, что считывание во второй раз продолжилось, после первого слова, а не с начала, так как первое слово было прочитано ранее.

Программа сработала правильно, но не всегда так бывает, даже в том случае, если с кодом всё в порядке. Например, в программу передано имя несуществующего файла или в имени допущена ошибка. Что тогда? В этом случае ничего не произойдёт вообще. Файл не будет найден, а значит и прочитать его невозможно. Поэтому компилятор проигнорирует строки, где

В результате корректно завершится работа программы, но ничего на экране показано не будет. Казалось бы, это вполне нормальная реакции на такую ситуацию. Но простому пользователю не будет понятно, в чём дело и почему на экране не появилась строка из файла. Так вот, чтобы всё было предельно понятно, в C++ предусмотрена такая функция — is_open(), которая возвращает целые значения: 1 — если файл был успешно открыт, 0 — если файл открыт не был. Доработаем программу с открытием файла, таким образом, чтобы если файл не открыт, выводилось соответствующее сообщение.

```
// file_read.cpp: определяет точку входа для консольного приложения.
 #include "stdafx.h"
#include <fstream>
#include <iostream>
using namespace std;
int main(int argc, char* argv[])
{
   setlocale(LC_ALL, "rus"); // корректное отображение кириллицы
   char buff[50]; // буфер промежуточного хранения считываемого из файла текста
   ifstream fin("cppstudio.doc"); // (ВВЕЛИ НЕ КОРРЕКТНОЕ ИМЯ ФАЙЛА)

   if (!fin.is_open()) // если файл не открыт
      cout << "Файл не может быть открыт!\n"; // сообщить об этом
   else
   {
   fin >> buff; // считали первое слово из файла
   cout << buff << endl; // напечатали это слово

   fin.getline(buff, 50); // считали строку из файла
   fin.close(); // закрываем файл
   cout << buff << endl; // напечатали эту строку
   }
   return 0;
}
```

Проверка потока и is_open()

Эта программа сообщит о невозможности открыть файл. Так что если программа работает с файлами, рекомендуется использовать эту функцию, is_open(), даже, если уверены, что файл существует.

Проверка потока и is_open()

Для файловых потоков is_open() включает в себя проверку успешности или неудачи операции открытия файла.

Проверка потока и is_open()

Например, попытка открытия для ввода несуществующего файла устанавливает флаг failbit. Поэтому можно было бы выполнить проверку следующим образом:

```
fin.open(argv[file]);
if (fin.failO) // попытка открытия не удалась
{
…..
}
```

Или же, поскольку объект ifstream, подобно istream, преобразуется в тип bool, когда ожидается именно этот тип, можно было бы использовать следующий код:

```
fin.open(argv[file]);
if (!fin) // попытка открытия не удалась
{
…
}
```

Проверка потока и is_open()

Это хороший способ проверки того, открыт ли файл — метод is_open ().

```
if (!fin.is_open ()) // попытка открытия не удалась
{
        …
}
```

Преимущество этого способа состоит в том, что он проверяет также наличие некоторых незначительных проблем, которые остаются незамеченными другими формами проверки.

Открытие нескольких файлов

Иногда может требоваться, чтобы программа открывала более одного файла. Стратегия открытия нескольких файлов зависит от того, как они будут использоваться. Если требуется, чтобы два файла были открыты одновременно, нужно создать отдельный поток для каждого файла. Например, программа, которая сравнивает два отсортированных файла и отправляет результат в третий, должна создать два объекта ifstream для двух входных файлов и один объект ofstream — для выходного файла. Количество файлов, которые можно открыть одновременно, зависит от операционной системы.

Однако можно запланировать последовательную обработку файлов. Предположим, что требуется подсчитать, сколько раз имя появляется в наборе из 10 файлов. В этом случае можно открыть единственный поток и по очереди ассоциировать его с каждым из этих файлов. При этом ресурсы компьютера используются экономнее, чем при открытии отдельного потока для каждого файла. Чтобы применить такой подход, нужно объявить объект ifstream без его инициализации, а затем с помощью метода open() ассоциировать поток с файлом.

Открытие нескольких файлов

Например, последовательное считывание двух файлов можно было бы организовать следующим образом:

```
ifstream fin; // создание потока конструктором по умолчанию
fin.open("fat.txt"); // ассоциирование потока с файлом fat.txt
...                  // выполнение каких-либо действий
fin.close (); // разрыв связи потока с файлом fat.txt
fin.clear(); // сброс fin (может не требоваться)
fin.open("rat.txt"); // ассоциирование потока с файлом rat.txt
…
fin.close();
```

Примеры работы с файлами

Пример 4. Заполнить файл значениями функции y = x * cos x.

/* Заполнить файл значениями функции y = x * cos x. */

```
/* Dev-C++ */
#include <cstdlib>
#include <iostream>
#include <fstream>
#include <cmath>

using namespace std;

double fun(double x);

int main()
{double a, b, h, x; char s[20];
 cout << "Enter the beginning and end of the segment, step-tabulation: ";
 cin >> a >> b >> h;
 cout << "File name? "; cin >> s;
 ofstream f;
 f.open(s);
 for (x=a; x<=b; x+=h)
  {f.width(10);   f << x;
   f.width(15);   f << fun(x) << endl;  }
 f.close();

    system("PAUSE");
    return EXIT_SUCCESS;
}
double fun(double x)
{ return x*cos(x); }
```

Примеры работы с файлами

Пример 5. Файл содержит несколько строк, в каждой из которых записано единственное выражение вида a#b (без ошибок), где a, b - целочисленные величины, # - операция +, -, /, *. Вывести каждое из выражений и их значения.

```
/* Dev-C++ */
#include <cstdlib>
#include <iostream>
#include <fstream>

using namespace std;
int main()
{
long a, b;  char s[256], c;  int i;
  cout << "File name? "; cin >> s;
  ifstream f;  f.open(s);
  while (!f.eof())
   {    f.getline(s, 256);
    i=0; a=0;
    while (s[i]>='0'&&s[i]<='9')
     {
         a=a*10+s[i]-'0';
          i++;
     }
    c=s[i++];    b=0;
    while (s[i]>='0'&&s[i]<='9')
     {
         b=b*10+s[i]-'0';
          i++;
     }
    switch (c){
    case '+': a+=b; break;
    case '-': a-=b; break;
    case '/': a/=b; break;
    case '*': a*=b; break;}
    cout << s << " = " << a << endl;   }
    f.close();
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Пример 6. В заданном файле целых чисел посчитать количество компонент, кратных 3.

```
/* В заданном файле целых чисел посчитать количество компонент, кратных 3. */
/* Dev-C++ */
#include <cstdlib>
#include <iostream>
#include <fstream>

using namespace std;
int main()
{int r,ch;
  ifstream f;
  f.open("CH_Z.TXT");
  ch=0;
  for (;f.peek()!=EOF;)
    {f>>r;
     cout << r << " ";
     if (r%3==0) ch++ ;
     }
f.close();
  cout << endl << "Answer: " << ch;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Литература

1. Файлы в C++. [Электронный ресурс]. Режим доступа: http://comp-science.narod.ru/Progr/file_c.htm
2. Работа с файлами в C++. [Электронный ресурс]. Режим доступа: http://cppstudio.com/post/446/
3. Бьярне Страуструп. Программирование. Принципы и практика с использованием C++. – М.: Вильямс, 2016. – 1328 с.
4. Стивен Язык программирования C++. Лекции и упражнения. – М.: Вильямс, 2017. – 1248 с.
5. Эндрю Кениг, Барбара Э. Му. Эффективное программирование на C++. Практическое программирование на примерах. - М.: Вильямс, 2016. – 368 с.
6. Алексей Васильев. Программирование на C++ в примерах и задачах. – М.: Эксмо, 2016. – 368 с.
7. Учебник по C++ для начинающих [Электронный ресурс]. Режим доступа: http://www.programmersclub.ru/main
8. Литвиненко Н. А. Технология программирования на C++ [Электронный ресурс] Режим доступа: http://www.proklondike.com/books/cpp/technology_of_programming_on_cplus.html
9. Стефан Р. Дэвис. C++ для чайников [Электронный ресурс] Режим доступа: http://www.proklondike.com/books/cpp/cplus_dlja_chainikov.html